TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT VĨNH LONG
KHOA CÔNG NGHỆ THÔNG TIN

GIÁO TRÌNH

LẬP TRÌNH CĂN BẢN

(BASIC PROGRAMMING)

BIÊN SOẠN
TH.S NGUYỄN VĂN HIẾU

Vĩnh Long - 2017

LỜI NÓI ĐẦU

-----ଓ✧ଌ-----

Để đáp ứng nhu cầu học tập và nghiên cứu của các bạn sinh viên, đặc biệt là sinh viên chuyên ngành Công nghệ thông tin, Khoa Công nghệ thông tin - Trường Đại học Sư phạm Kỹ thuật Vĩnh Long đã tiến hành biên soạn các giáo trình, bài giảng chính trong chương trình học. Giáo trình **Lập trình căn bản** này được biên soạn chủ yếu dựa trên quyển " ***C++ Program Design – An Introduction to Programming and Object-Oriented Design*** " của James P. Cohoon and Jack W.Davidson. Giáo trình này cũng được biên soạn dựa trên kinh nghiệm giảng dạy nhiều năm môn **Lập trình căn bản** của các giáo viên trong khoa chúng tôi. Ngoài ra chúng tôi cũng đã tham khảo rất nhiều tài liệu của các trường đại học trong và ngoài nước.

Tài liệu này được biên soạn dựa theo đề cương chi tiết môn học **Lập trình căn bản** của Khoa Công nghệ thông tin Trường Đại học Sư phạm Kỹ thuật Vĩnh Long dùng cho sinh viên chuyên ngành Công nghệ thông tin bao gồm 7 chương:

Chương 1: Tổng quan về ngôn ngữ C++

Chương 2: Các cấu trúc điều khiển

Chương 3: Dữ liệu kiểu mảng

Chương 4: Dữ liệu kiểu chuỗi

Chương 5: Con trỏ và hàm

Chương 6: Dữ liệu kiểu cấu trúc

Chương 7: Dữ liệu kiểu tập tin

Mục tiêu của nó nhằm giúp các bạn sinh viên chuyên ngành có một tài liệu cô đọng dùng làm tài liệu học tập và nghiên cứu nhưng chúng tôi cũng không loại trừ toàn bộ các đối tượng khác tham khảo. Chúng tôi nghĩ rằng các bạn sinh viên thuộc các chuyên ngành khác và những người quan tâm đến lập trình sẽ tìm được những điều bổ ích trong quyển giáo trình này.

Mặc dù chúng tôi đã rất cố gắng trong quá trình biên soạn nhưng chắc chắn giáo trình này sẽ còn nhiều thiếu sót và hạn chế. Rất mong nhận được sự đóng góp ý kiến quý báu của các sinh viên và bạn đọc để giáo trình ngày càng hoàn thiện hơn.

Chân thành cảm ơn!

Tác giả

CHƯƠNG 1: TỔNG QUAN VỀ NGÔN NGỮ C++

-----ଓ✧ଌ-----

1.1 KHÁI NIỆM VỀ LẬP TRÌNH

1.1.1 Giới thiệu chung

Máy tính là một công cụ để giải quyết hàng loạt các bài toán lớn. Một lời giải cho một bài toán nào đó được gọi là một giải thuật hay thuật toán (algorithm); nó mô tả một chuỗi các bước cần thực hiện để giải quyết bài toán.

Ví dụ: Bài toán về sắp xếp một danh sách các số theo thứ tự tăng dần.

Giải thuật của bài toán trên là: Giả sử danh sách đã cho là *list1*; ta tạo ra một danh sách rỗng *list2*, để lưu danh sách đã sắp xếp. Lặp đi lặp lại công việc, tìm số nhỏ nhất trong *list1*, xóa nó khỏi *list1* và thêm vào phần tử kế tiếp trong danh sách *list2*, cho đến khi *list1* là rỗng.

Giải thuật được diễn giải bằng các thuật ngữ trừu tượng mang tính chất dễ hiểu. Ngôn ngữ thật sự được hiểu bởi máy tính là ngôn ngữ máy. Chương trình được diễn đạt bằng ngôn ngữ máy được gọi là có thể thực thi. Một chương trình được viết bằng bất kỳ một ngôn ngữ nào trước hết cần được dịch sang ngôn ngữ máy để máy tính có thể hiểu và thực thi nó.

Ngôn ngữ máy cực kỳ khó hiểu đối với lập trình viên vì thế họ không thể sử dụng trực tiếp ngôn ngữ máy để viết chương trình. Một sự trừu tượng khác là ngôn ngữ assembly. Nó cung cấp những tên dễ nhớ cho các lệnh và một ký hiệu dễ hiểu hơn cho dữ liệu. Bộ dịch được gọi là assembler chuyển ngôn ngữ assembly sang ngôn ngữ máy.

Ngay cả những ngôn ngữ assembly cũng khó sử dụng. Những ngôn ngữ cấp cao như: C, C++, Pascal, … cung cấp các ký hiệu thuận tiện hơn nhiều cho việc lập trình cũng như thi hành các giải thuật.

Các ngôn ngữ cấp cao này giúp cho các lập trình viên không phải nghĩ nhiều về các ngôn ngữ cấp thấp vì thế giúp họ chỉ tập trung vào giải thuật. Trình biên dịch (compiler) sẽ đảm nhiệm việc dịch chương trình viết bằng ngôn ngữ cấp cao sang ngôn ngữ assembly. Mã assembly được tạo ra bởi trình biên dịch sau đó sẽ được tập hợp lại để cho ra một chương trình có thể thực thi.

1.1.2 Định nghĩa

Lập trình là kỹ thuật tổ chức dữ liệu và xây dựng quy trình xử lý cho máy tính làm việc thông qua ngôn ngữ lập trình chẳng hạn như: C, C++, Pascal, …

- Tổ chức dữ liệu: sắp xếp các thông tin nhằm phục vụ cho yêu cầu nào đó.

- Quy trình xử lý: bao gồm các chỉ thị để thực hiện các công việc như: tạo thông tin ban đầu, tính toán, sao chép, di chuyển, tìm kiếm, in kết quả, …

1.1.3 Giải thuật (Algorithm)

Muốn viết được một chương trình nào đó cho máy tính thực hiện điều trước tiên là ta phải tìm được giải thuật (thuật toán) cho công việc cần viết chương trình đó.

Giải thuật (thuật toán) là một tập hợp có thứ tự các bước tiến hành công việc nhằm đạt được kết quả mong muốn.

Ví dụ: Để viết được chương trình giải phương trình bậc 2 ($ax^2 + bx + c = 0$) ta cần phải tìm được các bước để thực hiện công việc này, đó chính là giải thuật:

Bước 1: Nhập 3 hệ số a, b và c

Bước 2: Tính $\Delta = b^2 - 4ac$

Bước 3: Xét dấu Δ

Nếu $\Delta < 0$ thì phương trình vô nghiệm

Ngược lại, nếu $\Delta = 0$ thì phương trình có nghiệm kép $x_1 = x_2 = -\frac{b}{2a}$

ngược lại ($\Delta > 0$) thì phương trình có 2 nghiệm phân biệt:

$$x_1 = \frac{-b + \sqrt{\Delta}}{2a} \; ; \; x_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

1.1.4 Đặc tính của giải thuật

- Phải kết thúc sau một số bước hữu hạn.
- Các bước trong giải thuật phải được máy chấp nhận và có thể thực hiện được.
- Xét hết tất cả các trường hợp có thể xảy ra.
- Áp dụng được cho tất cả các bài toán cùng loại (cùng dạng).

1.1.5 Các lưu ý

- Một bài toán có thể có nhiều giải thuật khác nhau.
- Một giải thuật được gọi là tốt nếu nó có các tính chất sau:
  + Đơn giản, dễ hiểu.
  + Tiết kiệm vùng nhớ.
  + Thời gian thực hiện nhanh.

1.1.6 Các công cụ thể hiện giải thuật

Để thể hiện giải thuật ta có thể dùng nhiều công cụ khác nhau, trong đó có 2 công cụ được dùng nhiều nhất là ngôn ngữ giả và lưu đồ.

1.1.6.1 Ngôn ngữ giả

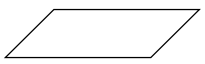
Ta có thể sử dụng các từ ngữ, ký hiệu sao cho ngắn gọn, dễ hiểu và có đánh số thứ tự các bước thực hiện.

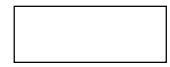
Ví dụ: Giải thuật giải phương trình bậc 2 ở trên được thể hiện bằng ngôn ngữ giả.

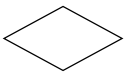
1.1.6.2 Lưu đồ (Flow chart)

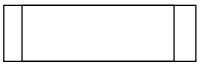
Ta sử dụng các khối theo qui định để thể hiện giải thuật, các khối được dùng là:

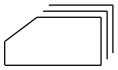

: Bắt đầu / Kết thúc

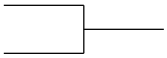

: Nhập / Xuất


: Thi hành (Thực hiện)


hoặc 
: Lựa chọn (Điều kiện)


: Chương trình con

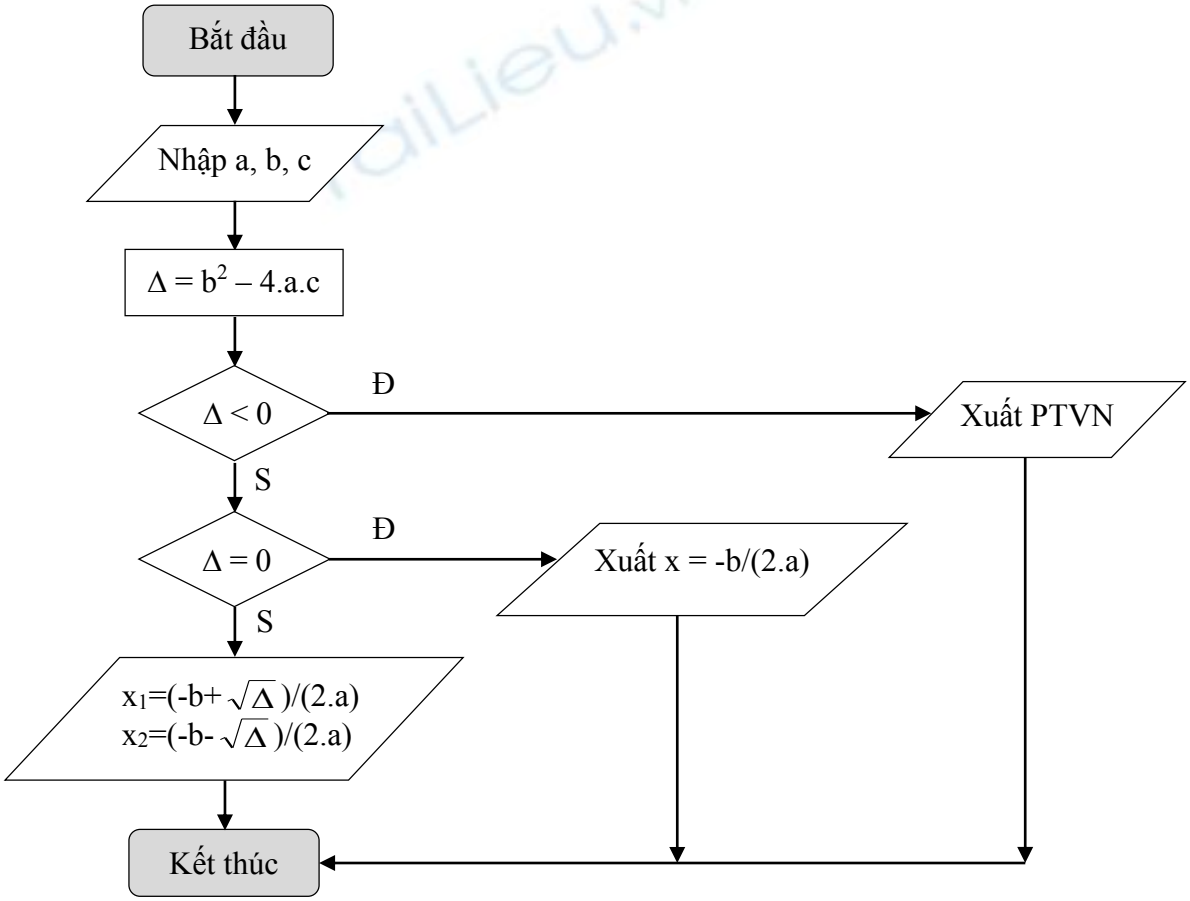

: Tập hợp các tập tin dữ liệu


: Các ghi chú, giải thích


: Nối

: Đường đi

Ví dụ: Giải thuật giải phương trình bậc 2 ở trên được thể hiện bằng lưu đồ như sau:



1.2 CÁC THÀNH PHẦN CƠ BẢN TRONG NGÔN NGỮ C++

Một ngôn ngữ lập trình (NNLT) cấp cao cho phép người sử dụng (NSD) biểu diễn ý tưởng của mình để giải quyết một vấn đề, một bài toán bằng cách diễn đạt gần với ngôn ngữ thông thường thay vì phải diễn đạt theo ngôn ngữ máy (dãy các ký hiệu 0, 1). Hiển nhiên, các ý tưởng NSD muốn trình bày phải được viết theo một cấu trúc chặt chẽ thường được gọi là giải thuật hay thuật toán và theo đúng các qui tắc của ngôn ngữ gọi là cú pháp hoặc văn phạm.

1.2.1 Bảng ký tự của C++

Dưới đây là bảng ký tự được dùng để tạo nên những câu lệnh của ngôn ngữ C++:

- Các chữ cái La tinh (viết thường và viết hoa): a..z và A..Z. Cùng một chữ cái nhưng viết thường phân biệt với viết hoa. Ví dụ chữ cái 'a' là khác với 'A'.

- Dấu gạch dưới: _

- Các chữ số thập phân: 0..9.

- Các ký hiệu toán học: +, -, *, /, % , &, ||, !, >, <, = ...

- Các ký hiệu đặc biệt khác: , ;: [ ], {}, #, dấu cách, ...

1.2.2 Từ khóa

Một từ khoá là một từ được qui định trước trong NNLT với một ý nghĩa cố định, thường dùng để chỉ các loại dữ liệu hoặc kết hợp thành câu lệnh. NSD có thể tạo ra những từ mới để chỉ các đối tượng của mình nhưng ***không được phép trùng với từ khóa***. Dưới đây là một vài từ khoá thường gặp, ý nghĩa của các từ này sẽ được trình bày dần trong các đề mục liên quan: asm, break, case, char, continue, default, do, double, else, extern, float, for, goto, if, int, long, register, return, short, sizeof, static, struct, switch, typedef, union, unsigned, while ... Một đặc trưng cần nhớ của C++ là ***các từ khóa luôn luôn được viết bằng chữ thường***.

1.2.3 Tên gọi

Để phân biệt các đối tượng với nhau chúng cần có một tên gọi. Hầu hết một đối tượng được viết ra trong chương trình thuộc 2 dạng, một dạng đã có sẵn trong ngôn ngữ (chẳng hạn như các từ khoá, tên các hàm chuẩn, ...), dạng còn lại là do NSD tạo ra dùng để đặt tên cho hằng, biến, kiểu, hàm, ...

Các tên gọi do NSD tự đặt phải tuân theo một số qui tắc sau:

- Là dãy ký tự liên tiếp (không chứa dấu cách và các ký hiệu) và phải bắt đầu bằng chữ cái hoặc dấu gạch dưới.

- Phân biệt ký tự in hoa và thường.

- Không được trùng với từ khóa.

- Số lượng ký tự dùng để phân biệt tên gọi là tùy ý.

Ví dụ:

Các tên gọi sau đây là đúng (được phép): i, i1, j, tinhoc, tin_hoc, luu_luong

Các tên gọi sau đây là sai (không được phép): 1i, tin hoc, luu-luong

Các tên gọi sau đây là khác nhau: ha_noi, Ha_noi, HA_Noi, HA_NOI

1.2.4 Chú thích trong chương trình

Một chương trình thường được viết một cách ngắn gọn, do vậy thông thường bên cạnh các câu lệnh chính thức của chương trình, NSD còn được phép viết vào chương trình các câu chú thích (ghi chú) nhằm giải thích rõ nghĩa hơn cho câu lệnh, đoạn chương

trình hay chương trình. Một chú thích có thể ghi chú về nhiệm vụ, mục đích, cách thức của thành phần đang được chú thích như biến, hằng, hàm hoặc công dụng của một đoạn lệnh, ... Các chú thích sẽ làm cho chương trình dễ hiểu hơn vì vậy dễ bảo trì, sửa chữa về sau hơn.

Có 2 cách báo cho chương trình biết một đoạn chú thích:

- Nếu chú thích là một đoạn ký tự bất kỳ liên tiếp nhau (trong 1 hàng hoặc trên nhiều hàng) ta đặt đoạn chú thích đó giữa cặp dấu đóng mở chú thích /* (mở) và */ (đóng).

- Nếu chú thích bắt đầu từ một vị trí nào đó cho đến hết hàng, thì ta đặt dấu // ở vị trí đó. Như vậy // sử dụng cho các chú thích chỉ trên 1 hàng.

Như đã nói ở trên, vai trò của đoạn chú thích là làm cho chương trình dễ hiểu đối với người đọc, vì vậy đối với máy các đoạn chú thích sẽ được bỏ qua. Lợi dụng đặc điểm này của chú thích đôi khi để tạm thời bỏ qua một đoạn lệnh nào đó trong chương trình (nhưng không xoá hẳn để khỏi phải gõ lại khi cần dùng đến) ta có thể đặt các dấu chú thích bao quanh đoạn lệnh này (ví dụ khi chạy thử chương trình, gỡ lỗi, ...), khi cần sử dụng lại ta có thể bỏ các dấu chú thích.

✧ ***Chú ý***: Cặp dấu chú thích /* ... */ không được phép viết lồng nhau.

1.2.5 Cấu trúc một chương trình trong C++

Một chương trình C++ có thể được đặt trong một hoặc nhiều file văn bản khác nhau. Mỗi file văn bản chứa một số phần nào đó của chương trình. Với những chương trình đơn giản và ngắn thường chỉ cần đặt chúng trên một file.

Một chương trình gồm nhiều hàm, mỗi hàm phụ trách một công việc khác nhau của chương trình. Đặc biệt trong các hàm này có một hàm duy nhất có tên hàm là main(). Khi chạy chương trình, các câu lệnh trong hàm main() sẽ được thực hiện đầu tiên. Trong hàm main() có thể có các câu lệnh gọi đến các hàm khác khi cần thiết, và các hàm này khi chạy lại có thể gọi đến các hàm khác nữa đã được viết trong chương trình (trừ việc gọi quay lại hàm main()). Sau khi chạy đến lệnh cuối cùng của hàm main() chương trình sẽ kết thúc.

Thông thường một chương trình gồm có các nội dung sau:

- *Phần khai báo các file nguyên mẫu:* Khai báo tên các file chứa những thành phần có sẵn (như các hằng chuẩn, kiểu chuẩn và các hàm chuẩn) mà NSD sẽ dùng trong chương trình.

- *Phần khai báo các kiểu dữ liệu, các biến, hằng ...*: Do NSD định nghĩa và được dùng chung trong toàn bộ chương trình.

- *Danh sách các hàm của chương trình*: Do NSD viết và bao gồm cả hàm main().

Ví dụ 1: Viết chương trình nhập vào số tuổi của bạn (t) và in ra màn hình hàng chuỗi: "Bây giờ bạn đã t tuổi."

```
#include <iostream>
using namespace std;
```

```
main()
{
      int t;
       cout << "Nhap vao so tuoi cua ban : ";
       cin >> t;
       cout << "Bay gio ban da "<<t<<" tuoi. "<<endl;
}
```

Hai hàng đầu tiên của chương trình là khai báo tiền xử lý là hai file nguyên mẫu `cstdlib` và `iostream`. Đây là khai báo bắt buộc vì trong chương trình có sử dụng các phương thức chuẩn "cout <<" (in ra màn hình) và "cin>>" (nhập dữ liệu từ bàn phím), các phương thức này được khai báo và định nghĩa sẵn trong `iostream`.

Không riêng hàm main(), mọi hàm khác đều phải bắt đầu tập hợp các câu lệnh của mình bởi dấu { và kết thúc bởi dấu }. Tập các lệnh bất kỳ bên trong cặp dấu này được gọi là khối lệnh. Khối lệnh là một cú pháp cần thiết trong các câu lệnh có cấu trúc như ta sẽ thấy trong các chương tiếp theo.

Ta cũng có thể sử dụng cách thức nhập và xuất của ngôn ngữ C bằng cách dùng phương thức scanf và printf. Ví dụ trên được viết lại như sau:

```
#include <stdio.h>
#include <conio.h>
main()
{
      int t;
       printf("Nhap vao so tuoi cua ban : ");
       scanf("%d",&t);
       printf("Bay gio ban da %d tuoi.\n",t);
       getch();
}
```

1.3 CÁC BƯỚC ĐỂ TẠO VÀ THỰC THI MỘT CHƯƠNG TRÌNH

1.3.1 Qui trình viết và thực thi chương trình

Trước khi viết và chạy một chương trình thông thường chúng ta cần:

a. Xác định yêu cầu của chương trình. Nghĩa là xác định dữ liệu đầu vào (input) cung cấp cho chương trình và tập các dữ liệu cần đạt được tức đầu ra (output). Các tập hợp dữ liệu này ngoài các tên gọi còn cần xác định kiểu của nó. Ví dụ để giải một phương trình bậc 2 dạng: $ax^2 + bx + c = 0$, cần báo cho chương trình biết dữ liệu đầu vào là a, b, c và đầu ra là nghiệm x_1 và x_2 của phương trình. Kiểu của a, b, c, x_1, x_2 là các số thực.

b. Xác định giải thuật.

c. Cụ thể hóa các khai báo kiểu và thuật toán thành dãy các lệnh. Tức là viết thành chương trình thông thường là trên giấy, sau đó bắt đầu soạn thảo vào trong máy. Quá trình này được gọi là soạn thảo chương trình nguồn.

d. Dịch chương trình nguồn để tìm và sửa các lỗi gọi là lỗi cú pháp.

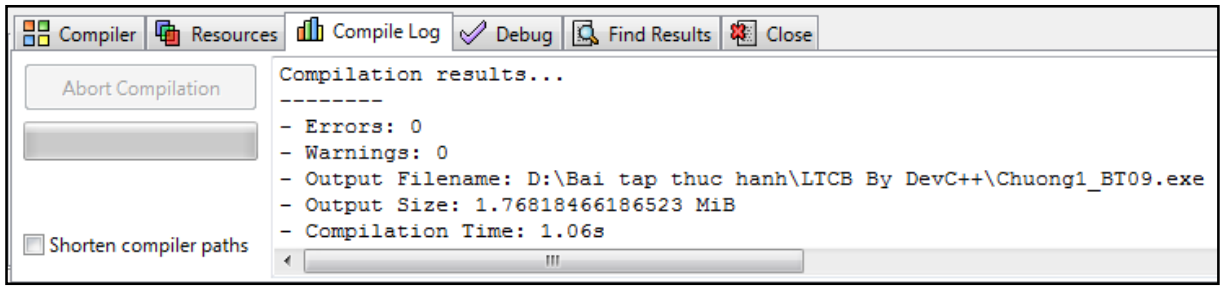
e. Chạy chương trình, kiểm tra kết quả in ra trên màn hình. Nếu sai, sửa lại chương trình, dịch và chạy lại để kiểm tra. Quá trình này được thực hiện lặp đi lặp lại cho đến khi chương trình chạy tốt theo yêu cầu đề ra của NSD.

1.3.2 Soạn thảo tập tin chương trình nguồn

Soạn thảo chương trình nguồn là một công việc gõ nội dung của chương trình (đã viết ra giấy) vào trong máy. Có thể soạn chương trình nguồn trên các bộ soạn thảo (editor) khác nhưng phải chạy trong môi trường tích hợp C++ (Dev-C++, Borland C++, Turbo C). Mục đích của soạn thảo là tạo ra một văn bản chương trình và đưa vào bộ nhớ của máy. Văn bản chương trình cần được trình bày rõ ràng, dễ hiểu. Các câu lệnh cần gióng thẳng cột theo cấu trúc của lệnh (các lệnh chứa trong một lệnh cấu trúc được trình bày thụt vào trong so với điểm bắt đầu của lệnh). Các chú thích nên ghi ngắn gọn, rõ nghĩa và phù hợp.

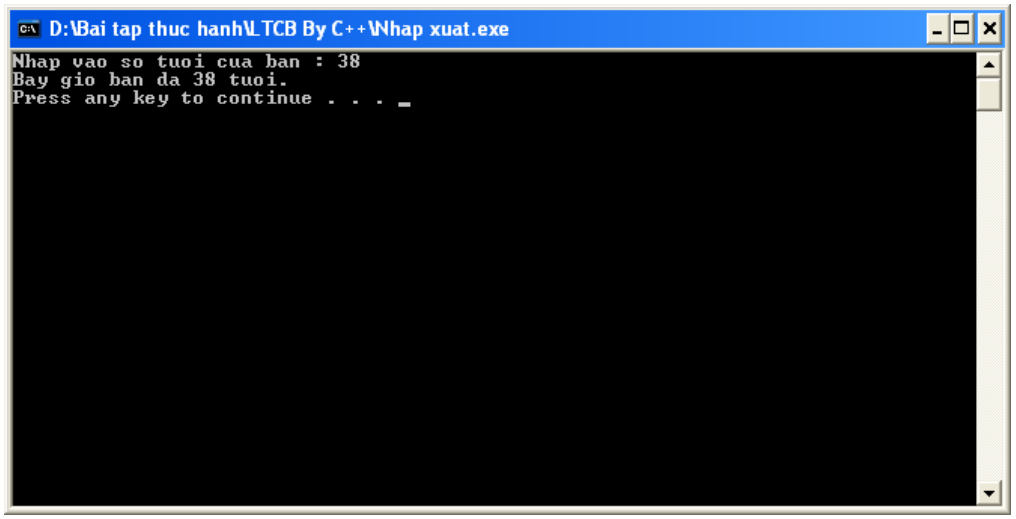
1.3.3 Dịch chương trình

Trong Dev-C++, sau khi đã soạn thảo xong chương trình nguồn, bước tiếp theo thường là dịch (vào menu Execute rồi chọn Compile hoặc ấn tổ hợp phím Ctrl-F9) để tìm và sửa các lỗi gọi là lỗi cú pháp. Trong khi dịch C++ sẽ đặt con trỏ vào nơi gây lỗi (viết sai cú pháp) trong văn bản. Sau khi sửa xong một lỗi NSD có thể chọn tiếp các lỗi khác để sửa tiếp ở cửa sổ Compiler. Quá trình sửa lỗi − dịch được lặp lại cho đến khi văn bản đã được sửa hết lỗi cú pháp. Sản phẩm sau khi dịch là một tệp mới gọi là chương trình đích có đuôi EXE tức là tệp mã máy để thực hiện.Tệp này có thể lưu tạm thời trong bộ nhớ phục vụ cho quá trình chạy chương trình hoặc lưu lại trên đĩa tùy theo tùy chọn khi dịch của NSD. Trong và sau khi dịch, C++ sẽ hiện một cửa sổ chứa thông báo về các lỗi (nếu có), hoặc xuất hiện hộp thoại thông báo chương trình đã được dịch thành công.



1.3.4 Thực thi chương trình

Sau khi đã sửa hết lỗi trong chương trình ta có thể cho chương trình thực thi (vào menu Execute rồi chọn Run hoặc ấn tổ hợp phím Ctrl-F10), nếu chương trình chưa dịch sang mã máy, máy sẽ tự động dịch lại trước khi chạy. Kết quả của chương trình sẽ hiện ra trong một cửa sổ kết quả để NSD kiểm tra.



Nếu kết quả chưa được như mong muốn, quay lại chương trình nguồn để chỉnh sửa và chạy lại chương trình để kiểm tra kết quả. Quá trình này được lặp lại cho đến khi chương trình chạy đúng như yêu cầu đã đề ra. Khi chương trình đang chạy, cửa sổ kết quả sẽ hiện ra tạm thời che khuất cửa sổ soạn thảo. Sau khi kết thúc chạy chương trình cửa sổ soạn thảo sẽ tự động hiện ra trở lại và che khuất cửa sổ kết quả.

1.4 NHẬP XUẤT TRONG C++

Trong phần này chúng ta làm quen một số lệnh đơn giản cho phép NSD nhập dữ liệu vào từ bàn phím hoặc xuất kết quả ra màn hình.

1.4.1 Nhập dữ liệu từ bàn phím

Để nhập dữ liệu vào cho các biến có tên bien_1, bien_2, bien_3 chúng ta sử dụng câu lệnh:

cin >> bien _1;

cin >> bien _2;

cin >> bien _3;

hoặc:

cin >> bien _1 >> bien _2 >> bien _3;

bien_1, bien_2, bien_3 là các biến được sử dụng để lưu trữ các giá trị NSD nhập vào từ bàn phím. Khái niệm biến sẽ được mô tả cụ thể hơn trong phần sau, ở đây bien_1, bien_2, bien_3 được hiểu là các tên gọi để chỉ 3 giá trị khác nhau. Hiển nhiên có thể nhập dữ liệu nhiều hơn 3 biến bằng cách tiếp tục viết tên biến vào bên phải sau dấu >> của câu lệnh.

Khi chạy chương trình nếu gặp các câu lệnh trên chương trình sẽ "tạm dừng" để chờ NSD nhập dữ liệu vào cho các biến. Sau khi NSD nhập xong dữ liệu, chương trình sẽ tiếp tục chạy từ câu lệnh tiếp theo sau của các câu lệnh trên.

Cách thức nhập dữ liệu của NSD phụ thuộc vào loại giá trị của biến cần nhập mà ta gọi là kiểu, ví dụ nhập một số có cách thức khác với nhập một chuỗi ký tự.

Giả sử cần nhập độ dài hai cạnh của một hình chữ nhật, trong đó cạnh dài được qui ước bằng tên biến cd và chiều rộng được qui ước bởi tên biến cr. Câu lệnh nhập sẽ như sau:

```
cin >> cd >> cr ;
```

Khi máy dừng chờ nhập dữ liệu NSD sẽ gõ giá trị cụ thể của các chiều dài, rộng theo đúng thứ tự trong câu lệnh. Các giá trị này cần cách nhau bởi ít nhất một dấu trắng (ta qui ước gọi dấu trắng là một trong 3 loại dấu được nhập bởi các phím sau: phím spacebar (dấu cách), phím tab (dấu tab) hoặc phím Enter (dấu xuống hàng)). Các giá trị NSD nhập vào cũng được hiển thị trên màn hình để NSD dễ theo dõi.

Ví dụ: Nếu NSD nhập vào 23 11 ↵ thì chương trình sẽ gán giá trị 23 cho biến cd và 11 cho biến cr.

✧ ***Chú ý***: Giả sử NSD nhập 23 và 11 không có dấu cách thì chương trình sẽ xem 23 và 11 là một giá trị và gán cho cd. Máy sẽ tạm dừng chờ NSD nhập tiếp giá trị cho biến cr.

1.4.2 Xuất dữ liệu ra màn hình

Để xuất/in giá trị của các biểu thức ra màn hình ta dùng câu lệnh sau:

```
cout << bt_1;
cout << bt_2;
cout << bt_3;
```

hoặc:

```
cout << bt_1 << bt_2 << bt_3;
```

Cũng giống câu lệnh nhập ở đây chúng ta cũng có thể mở rộng lệnh xuất/in với nhiều hơn 3 biểu thức. Câu lệnh trên cho phép in giá trị của các biểu thức bt_1, bt_2, bt_3. Các giá trị này có thể là tên của biến hoặc các kết hợp tính toán trên biến.

Ví dụ: Để in câu "Chiều dài là " và số 23 và tiếp theo là chữ " mét", ta có thể sử dụng 3 lệnh sau đây:

```
cout << "Chiều dài là " ;
cout << 23 ;
cout << " mét";
```

hoặc có thể chỉ bằng 1 lệnh:

```
cout << "Chiều dài là 23 mét" ;
```

Trường hợp chưa biết giá trị cụ thể của chiều dài, chỉ biết hiện tại giá trị này đã được lưu trong biến cd (ví dụ đã được nhập vào là 23 từ bàn phím bởi câu lệnh cin >> cd trước đó) và ta cần biết giá trị này là bao nhiêu thì có thể sử dụng câu lệnh in ra màn hình.

```
cout << "Chiều dài là " << cd << " mét" ;
```

Khi đó trên màn hình sẽ hiện ra hàng chữ: "Chiều dài là 23 mét". Như vậy trong trường hợp này ta phải dùng đến ba lần dấu phép toán << chứ không phải một như câu lệnh trên. Ngoài ra phụ thuộc vào giá trị hiện được lưu trong biến cd, chương trình sẽ in ra số chiều dài thích hợp chứ không chỉ in cố định thành "Chiều dài là 23 mét".

Ví dụ nếu cd được nhập là 15 thì lệnh trên sẽ in câu "Chiều dài là 15 mét".

Một giá trị cần in không chỉ là một biến như cd, cr, ... mà còn có thể là một biểu thức, điều này cho phép ta dễ dàng yêu cầu máy xuất ra diện tích và chu vi của hình chữ nhật khi đã biết cd và cr bằng các câu lệnh sau:

```
cout << "Dien tich = " << cd * cr ;
cout << "Chu vi = " << 2 * (cd + cr) ;
```

hoặc gộp tất cả thành 1 câu lệnh:

```
cout << "Dien tich = " << cd * cr << '\n' << " Chu vi = " << 2 * (cd + cr) ;
```

Ở đây có một ký tự đặc biệt: đó là ký tự '\n' ký hiệu cho ký tự xuống hàng (ta cũng có thể sử dụng cout<<endl), khi gặp ký tự này chương trình sẽ in các phần tiếp theo ở đầu hàng kế tiếp. Do đó kết quả của câu lệnh trên là 2 hàng sau đây trên màn hình:

```
Dien tich = 253
Chu vi = 68
```

Ở đây 253 và 68 lần lượt là các giá trị mà máy tính được từ các biểu thức cd * cr và 2 * (cd + cr) trong câu lệnh in ở trên.

✧ ***Chú ý***: Để sử dụng các câu lệnh nhập và xuất trong phần này, đầu chương trình phải có hàng khai báo

```
#include <iostream>
using namespace std;
```

Thông thường ta hay sử dụng lệnh in để in câu thông báo nhắc NSD nhập dữ liệu trước khi có câu lệnh nhập. Khi đó trên màn hình sẽ hiện hàng thông báo này rồi mới tạm dừng chờ dữ liệu nhập vào từ bàn phím. Nhờ vào thông báo này NSD sẽ biết phải nhập dữ liệu, nhập nội dung gì và như thế nào, ...

Ví dụ:

```
cout << "Hay nhap chieu dai: ";
cin >> cd;
cout << "Hay nhap chieu rong: ";
cin >> cr;
```

Khi đó máy sẽ in hàng thông báo "Hay nhap chieu dai: " và chờ sau khi NSD nhập xong 23 ↵, máy sẽ thực hiện câu lệnh tiếp theo tức in hàng thông báo "Hay nhap chieu rong: " và chờ đến khi NSD nhập xong 11 ↵ chương trình sẽ tiếp tục thực hiện các câu lệnh tiếp theo.

Ví dụ 2: Từ các thảo luận trên ta có thể viết một cách đầy đủ chương trình tính diện tích và chu vi của một hình chữ nhật. Để chương trình có thể tính với các bộ giá trị khác nhau của chiều dài và rộng ta cần lưu giá trị này vào trong các biến chẳng hạn như cd và cr.

```
#include <iostream>
using namespace std;
main()
{
    float cd, cr ;
    cout << "Hay nhap chieu dai: " ;
    cin >> cd ;
    cout << "Hay nhap chieu rong: " ;
    cin >> cr ;
    cout << "Dien tich = " << cd * cr << '\n' ;
    cout << "Chu vi = " << 2 * (cd + cr) << '\n';
}
```

Khi chạy đến câu lệnh nhập, chương trình dừng để chờ nhận chiều dài và chiều rộng, NSD nhập các giá trị cụ thể, chương trình sẽ tiếp tục thực hiện và in ra kết quả. Thông qua câu lệnh nhập dữ liệu và 2 biến cd, cr NSD có thể yêu cầu chương trình cho kết quả của một hình chữ nhật bất kỳ chứ không chỉ trong trường hợp hình có chiều dài 23 và chiều rộng 11 như trong ví dụ cụ thể trên.

1.4.3 Định dạng thông tin cần in ra màn hình

Một số định dạng đơn giản được trình bày trước ở đây. Các định dạng chi tiết và phức tạp hơn sẽ được trình bày trong các phần sau của giáo trình. Để sử dụng các định dạng này cần khai báo file nguyên mẫu <iomanip.h> ở đầu chương trình bằng chỉ thị `#include <iomanip>`

- **endl:** Tương đương với ký tự xuống hàng '\n'.

- **setw(n):** Bình thường các giá trị được in ra bởi lệnh cout << sẽ thẳng theo lề trái với độ rộng phụ thuộc vào độ rộng của giá trị đó. Phương thức này qui định độ rộng dành để in ra các giá trị là n cột màn hình. Nếu n lớn hơn độ dài thực của giá trị, giá trị sẽ in ra theo lề phải, để trống phần thừa (dấu cách) ở trước.

- **setprecision(n):** Chỉ định số chữ số thể hiện là n. Số sẽ được làm tròn trước khi in ra.

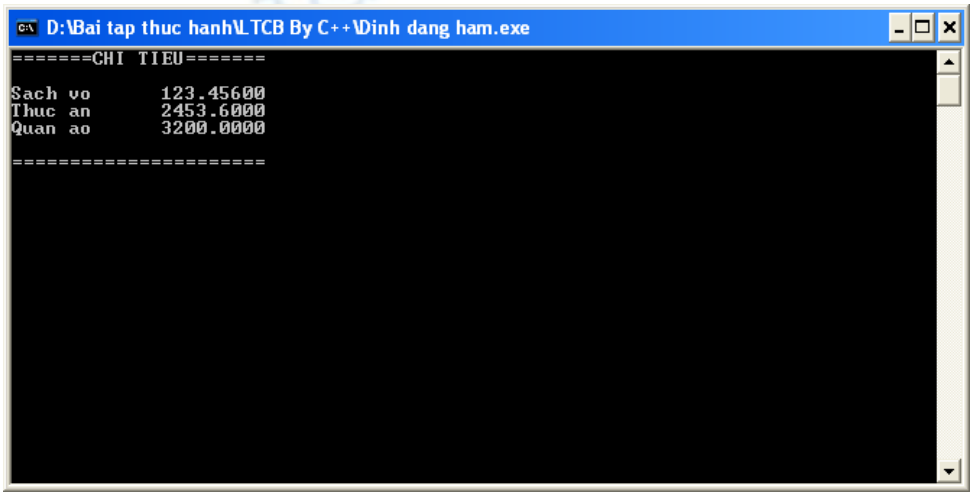
- **setiosflags(ios::showpoint):** Phương thức setprecision chỉ có tác dụng trên một hàng in. Để cố định các giá trị đã đặt cho mọi hàng in (cho đến khi đặt lại giá trị mới) ta sử dụng phương thức setiosflags(ios::showpoint).

Ví dụ 3: Viết chương trình in có định dạng mức chi tiêu của một sinh viên.

```
#include <iostream>
#include <iomanip>
#include <conio.h> // De su dung ham getch()
using namespace std;
```

```
main()
{
    cout << "=======" << "CHI TIEU" << "=======" << endl ;
    cout<<endl;
    cout << setiosflags(ios::showpoint) << setprecision(8) ;
    cout << "Sach vo" << setw(15) << 123.456 << endl;
    cout << "Thuc an" << setw(15) << 2453.6 << endl;
    cout << "Quan ao" << setw(15) << 3200.0 << endl;
    cout <<endl;
    cout << "======================" << endl ;
    getch();
}
```

Chương trình này khi chạy sẽ in ra kết quả như sau:



1.4.4 Một số lưu ý

Toán tử nhập >> chủ yếu làm việc với dữ liệu kiểu số. Để nhập ký tự hoặc chuỗi (chuỗi) ký tự, C++ cung cấp các phương thức (hàm) sau:

- **cin.get(c):** Cho phép nhập một ký tự vào biến ký tự c.

- **cin.getline(s, n):** Cho phép nhập tối đa n-1 ký tự vào chuỗi s.

Các hàm trên khi thực hiện sẽ lấy các ký tự còn lại trong bộ nhớ đệm (của lần nhập trước) để gán cho c hoặc s. Do toán tử cin >> x sẽ để lại ký tự xuống hàng trong bộ đệm nên ký tự này sẽ làm trôi các lệnh sau đó như cin.get(c), cin.getline(s,n) (máy không dừng để nhập cho c hoặc s).

Vì vậy trước khi sử dụng các phương thức cin.get(c) hoặc cin.getline(s, n) ta nên sử dụng phương thức cin.ignore(1) để lấy ra ký tự xuống hàng còn sót lại trong bộ đệm.

- **gets(st):** cho phép nhập vào một chuỗi ký tự tùy ý cho biến st.

Ví dụ 4: Viết chương trình nhập lần lượt giá trị cho 4 biến kiểu số nguyên, ký tự, số thực và chuỗi. Sau đó in ra kết quả giá trị của các biến đó.

```
#include <iostream>
#include <conio.h>
using namespace std;
main()
{
    int x; float y;
    char c; char st[20];
    cout << "Nhap so nguyen x = "; cin >> x;
    cin.ignore(1);
    cout << "Nhap ky tu c = "; cin.get(c);
    cout << "Nhap so thuc y = "; cin >> y;
    cin.ignore(1);
    cout << "Nhap chuoi st = "; gets(st);
    cout << "Ket qua cua cac bien da nhap la:" << endl;
    cout << "So nguyen x = " << x << endl;
    cout << "Ky tu c = " << c << endl;
    cout << "So thuc y = " << y << endl;
    cout << "Chuoi st = " << st << endl;
}
```

1.5 CÁC KIỂU DỮ LIỆU CƠ BẢN

1.5.1 Khái niệm về kiểu dữ liệu

Thông thường dữ liệu hay dùng là số và chữ. Tuy nhiên việc phân chia chỉ 2 loại dữ liệu như thế là không đủ. Để dễ dàng hơn cho lập trình, hầu hết các NNLT đều phân chia dữ liệu thành nhiều kiểu khác nhau được gọi là các kiểu cơ bản hay kiểu chuẩn. Trên cơ sở kết hợp các kiểu dữ liệu chuẩn, NSD có thể tự đặt ra các kiểu dữ liệu mới để phục vụ cho chương trình giải quyết bài toán của mình. Có nghĩa là lúc đó mỗi đối tượng được quản lý trong chương trình sẽ là một tập hợp nhiều thông tin hơn và được tạo thành từ nhiều loại (kiểu) dữ liệu khác nhau.

Một biến là một số ô nhớ liên tiếp nào đó trong bộ nhớ dùng để lưu trữ dữ liệu (vào, ra hay kết quả trung gian) trong quá trình hoạt động của chương trình. Để quản lý chặt chẽ các biến, NSD cần khai báo cho chương trình biết trước tên biến và kiểu của dữ liệu được chứa trong biến. Việc khai báo này sẽ làm chương trình quản lý các biến dễ dàng hơn

như trong việc cấp phát bộ nhớ cũng như quản lý các tính toán trên biến theo nguyên tắc: chỉ có các dữ liệu cùng kiểu với nhau mới được phép làm toán với nhau. Do đó, khi đề cập đến một kiểu chuẩn của một NNLT, thông thường chúng ta sẽ xét đến các yếu tố sau:

- ***Tên kiểu***: là một từ dành riêng để chỉ định kiểu của dữ liệu.

Ví dụ: Kiểu số nguyên có tên là `int`.

- ***Kích thước***: là số byte vùng nhớ cần thiết để lưu trữ một đơn vị dữ liệu thuộc kiểu này. Thông thường số byte này phụ thuộc vào các trình biên dịch và hệ thống máy tính khác nhau, ở đây ta chỉ xét đến hệ thống máy PC thông dụng hiện nay.

Ví dụ: Kiểu số nguyên `(int)` có kích thước là 2 bytes.

- ***Miền giá trị***: là tập giá trị mà một biến thuộc kiểu dữ liệu này sẽ có thể nhận được, người ta thường ghi giá trị nhỏ nhất và lớn nhất là bao nhiêu. Hiển nhiên các giá trị này phụ thuộc vào số byte mà hệ thống máy tính qui định cho từng kiểu. NSD cần nhớ đến miền giá trị này để khai báo kiểu cho các biến cần sử dụng một cách thích hợp.

Ví dụ: Kiểu số nguyên `(int)` có miền giá trị là -32768 .. 32767.

Bảng 1.1 sau đây mô tả tóm tắt một số kiểu dữ liệu chuẩn được sử dụng trong ngôn ngữ lập trình C++

LOẠI DỮ LIỆU	TÊN KIỂU	KÍCH THƯỚC	MIỀN GIÁ TRỊ
Luận lý	bool	1 byte	false, true
Ký tự	[signed] char	1 byte	-128 .. 127
	unsigned char	1 byte	0 .. 255
Số nguyên	int	2 bytes	-32768 .. 32767
	unsigned int	2 bytes	0 .. 65535
	short	2 bytes	-32768 .. 32767
	long [int]	4 bytes	-2^{31} .. $2^{31} - 1$
	unsigned long	4 bytes	0 .. $2^{32} - 1$
Số thực	float	4 bytes	$3.4*10^{-38}$ .. $3.4*10^{+38}$
	double	8 bytes	$1.7*10^{-308}$ .. $1.7*10^{+308}$
	long double	10 bytes	$3.4*10^{-4932}$ .. $1.1*10^{+4932}$

1.5.2 Kiểu luận lý

Kiểu luận lý còn được gọi là kiểu logic (bool) chỉ gồm 2 giá trị là false (sai) và true (đúng). Kết quả của một biểu thức điều kiện hay biểu thức so sánh luôn là kiểu luận lý.

Trong ngôn ngữ lập trình trình C chuẩn và các phiên bản trước của Dev-C++ không có kiểu bool, khi đó để thể hiện giá trị false và true người ta sử dụng kiểu số nguyên (int) với giá trị bằng 0 tương ứng là false và giá trị khác 0 tương ứng là true.

1.5.3 Kiểu ký tự

Một ký tự là một ký hiệu trong bảng mã ASCII. Như đã biết một số ký tự có mặt chữ trên bàn phím (ví dụ các chữ cái, chữ số) trong khi một số ký tự lại không (ví dụ ký tự biểu diễn việc lùi lại một ô trong văn bản, ký tự chỉ việc kết thúc một hàng hay kết thúc một văn bản).

Do vậy để biểu diễn một ký tự người ta dùng chính mã ASCII của ký tự đó trong bảng mã ASCII và thường gọi là giá trị của ký tự. Ví dụ phát biểu "Cho ký tự 'A'" là tương đương với phát biểu "Cho ký tự 65" (65 là mã ASCII của ký tự 'A') hay "Xóa ký tự xuống hàng" là tương đương với phát biểu "Xóa ký tự 13" vì 13 là mã ASCII của ký tự xuống hàng. Như vậy một biến kiểu ký tự có thể được nhận giá trị theo 2 cách tương đương là chữ hoặc số.

Ví dụ: Giả sử c là một biến ký tự thì câu lệnh gán c = 'A' cũng tương đương với câu lệnh gán c = 65. Tuy nhiên để sử dụng giá trị số của một ký tự c nào đó ta phải yêu cầu đổi c sang giá trị số bằng câu lệnh int(c).

1.5.4 Kiểu số nguyên

Các số nguyên được phân chia thành 4 loại kiểu khác nhau với các miền giá trị tương ứng được cho trong bảng 1.1. Đó là kiểu số nguyên ngắn (short) tương đương với kiểu số nguyên (int) sử dụng 2 bytes và số nguyên dài (long int hoặc có thể viết gọn hơn là long) sử dụng 4 bytes. Kiểu số nguyên thường được chia làm 2 loại có dấu (int) và không dấu (unsigned int hoặc có thể viết gọn hơn là unsigned).

Ta thường sử dụng kiểu int cho các số nguyên trong các bài toán với miền giá trị vừa phải chẳng hạn như các biến đếm trong các vòng lặp, năm sinh của một người, ...

1.5.5 Kiểu số thực

Để sử dụng số thực ta cần khai báo kiểu float hoặc double mà miền giá trị của chúng được cho trong bảng 1.1. Các giá trị số kiểu double được gọi là số thực với độ chính xác kép vì với kiểu dữ liệu này máy tính có cách biểu diễn khác so với kiểu float để đảm bảo số số lẻ sau một số thực có thể tăng lên đảm bảo tính chính xác cao hơn so với số kiểu float. Tuy nhiên, trong các bài toán thông dụng thường ngày độ chính xác của số kiểu float là đủ.

Như đã nhắc đến trong phần nhập/xuất, liên quan đến việc in ấn số thực ta có một vài cách thiết lập dạng in theo ý muốn, ví dụ độ rộng tối thiểu để in một số hay số chữ số cần in là bao nhiêu, ...

Ví dụ 5: Chương trình sau đây sẽ in diện tích và chu vi của một hình tròn có bán kính 3cm với 6 số chữ số được in ra.

```
#include <iostream>
#include <conio.h>
#include <iomanip>
using namespace std;
main()
{
    float r = 3;
    cout<< setiosflags(ios::showpoint);
    cout<< "DT = "<<setprecision(6)<<r*r*3.1416<<endl;
    cout<< "CV = "<<setprecision(6)<<2*3.1416*r<<endl;
}
```

1.6 HẰNG SỐ

1.6.1 Định nghĩa

Hằng số (constant) là một giá trị cố định trong suốt quá trình thực thi chương trình.

Ví dụ:

3 là hằng số nguyên

'A' là hằng ký tự

5.0 là hằng số thực

"Viet Nam" là hằng chuỗi ký tự

Một giá trị có thể được hiểu dưới nhiều kiểu khác nhau, do vậy khi viết hằng ta cũng cần có dạng viết thích hợp.

1.6.2 Một số hằng thông dụng

Đối với một số hằng ký tự thường dùng nhưng không có mặt chữ tương ứng, hoặc các ký tự được dành riêng với nhiệm vụ khác, khi đó thay vì phải nhớ giá trị của chúng ta có thể viết theo qui ước sau:

'\n'	:	biểu thị ký tự xuống hàng (cũng tương đương với endl)
'\t'	:	ký tự tab
'\a'	:	ký tự chuông (tức thay vì in ký tự, loa sẽ phát ra một tiếng 'bíp')
'\r'	:	xuống hàng
'\f'	:	kéo trang
'\\'	:	dấu \
'\?'	:	dấu chấm hỏi ?

'\"' : dấu nháy đơn '

'\"' : dấu nháy kép "

'\kkk' : ký tự có mã là kkk trong hệ 8

'\xkk' : ký tự có mã là kk trong hệ 16

Ví dụ: `cout << "Hom nay troi \t nang \a \a \a \n" ;` sẽ in ra màn hình hàng chữ "Hôm nay trời" sau đó bỏ một khoảng cách bằng một tab (khoảng 8 dấu cách) rồi in tiếp chữ "nắng", tiếp theo phát ra 3 tiếng chuông và cuối cùng con trỏ trên màn hình sẽ nhảy xuống đầu hàng mới.

1.6.3 Khai báo

Một giá trị cố định (hằng) được sử dụng nhiều lần trong chương trình đôi khi sẽ thuận lợi hơn nếu ta đặt cho nó một tên gọi, thao tác này được gọi là khai báo hằng. Ví dụ một chương trình quản lý sinh viên với giả thiết số sinh viên tối đa là 50. Nếu số sinh viên tối đa không thay đổi trong chương trình ta có thể đặt cho nó một tên gọi như **sosv** chẳng hạn. Trong suốt chương trình bất kỳ chỗ nào xuất hiện giá trị 50 ta đều có thể thay nó bằng **sosv**.

Việc sử dụng tên hằng thay cho hằng có nhiều điểm thuận lợi như sau:

- Chương trình dễ đọc hơn, vì thay cho các con số ít có ý nghĩa, một tên gọi sẽ làm NSD dễ hình dung vai trò, nội dung của nó. Ví dụ, khi gặp tên gọi **sosv** NSD sẽ dễ hình dung "đây là số sinh viên tối đa trong một lớp", trong khi số 50 có thể là số sinh viên mà cũng có thể là số thứ tự của một sinh viên nào đó.

- Chương trình dễ sửa lỗi hơn, ví dụ bây giờ nếu muốn thay đổi chương trình sao cho bài toán quản lý được thực hiện với số sinh viên tối đa là 60, khi đó ta cần tìm và thay thế hàng trăm vị trí xuất hiện của 50 thành 60. Việc thay thế như vậy dễ gây ra lỗi vì có thể không tìm thấy hết các số 50 trong chương trình hoặc thay nhầm số 50 với ý nghĩa khác như số thứ tự của một sinh viên nào đó chẳng hạn. Nếu trong chương trình sử dụng hằng **sosv**, bây giờ việc thay thế trở nên chính xác và dễ dàng hơn bằng thao tác khai báo lại giá trị hằng **sosv** bằng 60. Lúc đó trong chương trình bất kỳ nơi nào gặp tên hằng **sosv** đều được chương trình hiểu với giá trị 60.

Để khai báo hằng ta dùng các câu khai báo sau:

#define tên_hằng giá_trị_hằng;

hoặc:

const <Kiểu> tên_hằng = giá_trị_hằng;

Ví dụ:

```
#define sosv 50;
#define MAX 100;
const int sosv = 50;
const float MAX = 100.0;
```

1.7 BIẾN

1.7.1 Định nghĩa

Biến (variable – biến số) là các tên gọi để lưu giá trị khi làm việc trong chương trình. Các giá trị được lưu có thể là các giá trị dữ liệu ban đầu, các giá trị trung gian tạm thời trong quá trình tính toán hoặc các giá trị kết quả cuối cùng. Khác với hằng, giá trị của biến có thể thay đổi trong quá trình làm việc bằng các lệnh đọc vào từ bàn phím hoặc lệnh gán. Hình ảnh cụ thể của biến là một số ô nhớ trong bộ nhớ được sử dụng để lưu các giá trị của biến.

1.7.2 Khai báo

Mọi biến phải được khai báo trước khi sử dụng. Một khai báo như vậy sẽ báo cho chương trình biết về một biến mới gồm có: tên của biến, kiểu của biến (tức là kiểu của giá trị dữ liệu mà biến sẽ lưu giữ). Thông thường với nhiều NNLT tất cả các biến phải được khai báo ngay từ đầu chương trình hay đầu của hàm, tuy nhiên để thuận tiện hơn cho người lập trình C++ cho phép khai báo biến ngay bên trong chương trình hoặc hàm, có nghĩa là bất kỳ lúc nào NSD thấy cần thiết sử dụng biến mới, họ có quyền khai báo và sử dụng nó từ đó trở đi.

Cú pháp khai báo biến gồm tên kiểu, tên biến và có thể có hay không khởi tạo giá trị ban đầu cho biến. Để khởi tạo hoặc thay đổi giá trị của biến ta dùng lệnh gán (=).

1.7.2.1 Khai báo không khởi tạo

tên_kiểu tên_biến_1;

tên_kiểu tên_biến_2;

tên_kiểu tên_biến_3;

Nhiều biến cùng kiểu có thể được khai báo trên cùng một hàng:

tên_kiểu tên_biến_1, tên_biến_2, tên_biến_3;

Ví dụ:

```
int i, j;
float x;
char c, d[100];
```

1.7.2.2 Khai báo có khởi tạo

Trong câu lệnh khai báo, các biến có thể được gán ngay giá trị ban đầu bởi phép toán gán (=) theo cú pháp:

tên_kiểu tên_biến_1 = gt_1;

tên_kiểu tên_biến_2 = gt_2,

tên_kiểu tên_biến_3 = gt_3;

Trong đó các giá trị gt_1, gt_2, gt_3 có thể là các hằng, biến hoặc biểu thức.